This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

A MEC-based Extended Virtual Sensing for Automotive Services

Giuseppe Avino, *Member, IEEE,* Paolo Bande, Pantelis A. Frangoudis, Christian Vitale, *Member, IEEE,* Claudio Casetti, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE,* Kalkidan Gebru, Adlen Ksentini, *Senior Member, IEEE*, Giuliana Zennaro

***Abstract*—Multi-access edge computing (MEC) comes with the promise of enabling low-latency applications and of reducing core network load by offloading traffic to edge service instances. Recent standardization efforts, among which the ETSI MEC, have brought about detailed architectures for the MEC. Leveraging the ETSI model, in this paper we first present a flexible, yet full-fledged, MEC architecture that is compliant with the standard specifications. We then use such architecture, along with the popular OpenAir Interface (OAI), for the support of automotive services with very tight latency requirements. We focus in particular on the Extended Virtual Sensing (EVS) services, which aim at enhancing the sensor measurements aboard vehicles with the data collected by the network infrastructure, and exploit this information to achieve better safety and improved passengers/driver comfort. For the sake of concreteness, we select the intersection control as an EVS service and present its design and implementation within the MEC platform. Experimental measurements obtained through our testbed show the excellent performance of the MEC EVS service against its equivalent cloud-based implementation, proving the need for MEC to support critical automotive services, as well as the benefits of the solution we designed.**

***Index Terms*—Softwarized networks, Automotive services, Road safety, Multi-access Edge Computing, V2I communications**

I. Introduction

Every year, the death toll caused by road accidents globally amounts to more than a million people, a worrying figure that is sadly complemented by almost 50 million people left with long-lasting or permanent injuries. The automotive sector has been actively engaged in finding solutions, first by equipping new cars with more sophisticated active safety systems such as the anti-lock braking system and electronic stability control, and more recently by using ICT at the core of its accident prevention technology. A specific example of the latter is the class of automotive services known as Extended Virtual Sensing (EVS), which leverages vehicular communication to collect the output of on-board vehicle sensors, merge them with smart city sensors, and distribute up-to-date information to increase the awareness of the surrounding environment.

In the area of vehicular communication, the consolidated standards by IEEE and ETSI have recently been challenged by the rise of 5G networks promising the delivery of one-stop solutions for integrated vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-pedestrian (V2P) operations. The deployment of 5G networks, expected by 2020, will offer a number of vertical industries (or, more simply, "verticals") a ready-to-use set of network services. Among the verticals, the automotive industry is looking at the capabilities of 5G networks in order to assess the potentiality of the C-V2X (Cellular Vehicle-to-Anything) technology for road safety. Among all the Key Performance Indicators (KPIs) of 5G that verticals, operators, and researchers alike hope to exploit, low latency and service portability are of paramount importance in view of devising effective EVS services. The cornerstone of the 5G architecture that enables these KPIs is the localized computational infrastructure represented by Multi-access Edge Computing (MEC).

In this paper, we look at intersection control as one of the premier EVS services, with the aim to reduce the risk of collision between vehicles. In order to assess the ability of softwarized networks to support EVS services, we present a *test-bed* implementation of such a road safety application on an OpenAir Interface (OAI) architecture including MEC functionalities, and evaluate the performance with a hardware-in-the loop simulation technique. With respect to prior art, our work contributes to the advancement of the field in multiple ways, as expounded below.

- **MEC systems:** our MEC implementation is fully compliant with ETSI MEC specifications; it covers all layers of the MEC reference architecture, as well as all of its main components and native services. There is currently no documented MEC system including all these features.
- **EVS services:** not only are the design of the EVS application and its implementation within the MEC platform novel contributions, but they also serve to demonstrate a full-fledged automotive service with distinctive and practically relevant characteristics. Specifically, (i) the service we develop integrates components that can be offered by diverse entities (transportation authorities, various automotive software vendors) in a microservice-based design; (ii) standard message formats are used for V2X communication; (iii) our design supports the deployment of diverse collision detection algorithms in a pluggable manner. We present one such algorithm and use it as a realistic and state-of-the-art collision detection mechanism. Our algorithm exhibits a complexity representative of this

A. Avino, C. Vitale, C. Casetti, C. F. Chiasserini, and K. Gebru are with Politecnico di Torino, Italy and CNIT, Italy. C. Vitale is also with KIOS Center of Excellence, Cyprus. C. F. Chiasserini is also with CNR-IEIIT and, along with C. Casetti, with CNIT, Italy. P. Bande and G. Zennaro are with CRF-FCA, Torino, Italy. P. Frangoudis and A. Ksentini are with EURECOM, Sophia Antipolis, France.

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

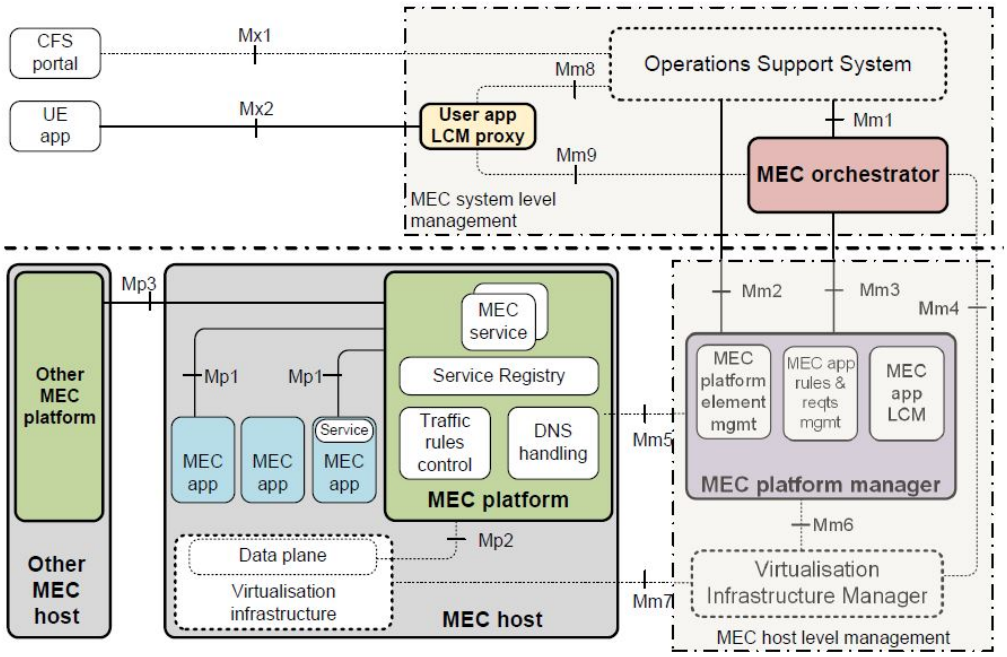

Fig. 1: ETSI MEC architecture [1]

type of applications, and perfectly serves to evaluate the suitability of MEC for automotive services with strict end-to-end latency requirements, accounting for realistic processing delays due to the algorithm execution.

- **Scalability:** our design features scaling capabilities to ensure that MEC applications can still meet their latency requirements as traffic load increases. Note that this is a very relevant contribution since we are the first to implement scalability using MEC standard-based reference points and interfaces. Also, our method is not limited to automotive services but it has a more generic applicability.
- **Evaluation of automotive services over MEC:** for the first time, we evaluate a full automotive service on top of a real, standard-compliant, MEC platform. This serves to demonstrate the suitability (and actually the need) of MEC, a core feature of the forthcoming 5G mobile communication systems, for the support of ultra-reliable and low-latency communications (URLLC) services as those required by automotive verticals.

II. Multi-access Edge Computing Architecture

MEC comes with the promise of enabling low-latency applications, exploiting distributed heterogeneous computing and network resources close to the user end, and reducing core network load by offloading traffic to edge service instances. Recent standardization efforts have brought about detailed architectures for MEC. The ETSI MEC Industry Specification Group (ISG) has provided a reference MEC architecture [2], specifying its components and their interfaces, as shown in Fig. 1. The main entities it includes are as follows:

- **MEC host**: It provides the execution environment to run (virtualized) Mobile Edge (ME) applications, and includes a MEC Platform (MEP) and a virtualization infrastructure, where ME applications are deployed.
- **MEP**: This component acts as the interface between the mobile network and ME applications. The MEP interacts with the mobile network over the (non-standardized) *Mp2* interface to access the user data plane, while it exposes MEC services via the *Mp1* reference point. The MEP Manager (MEPM) is responsible for MEP configuration and ME application lifecycle management, under the control of the Mobile Edge Orchestrator (MEO).
- **MEC services**: They are discovered and consumed by MEC applications over the Mp1 reference point. The ETSI MEC standards specify a set of MEC services that are provided natively by the MEC platform, as is the case for the Radio Network Information Service (RNIS) [3], local DNS, or traffic rules control. At the same time, via Mp1, third-party MEC applications can register and provide their own services. Note that not all MEC applications necessarily provide or consume MEC services.
- **Mobile Edge Orchestrator (MEO)**: It maintains a global view of the whole mobile edge network and is in charge of managing ME applications. The MEO is the interface between the Operations Support System/Business Support System (OSS/BSS) and the MEC platform and host. By interacting with the MEP and the virtual infrastructure, it supports the lifecycle management (e.g., instantiation and termination) of ME applications.

Further extensions of the MEC architecture towards network slicing [4], [5] are currently under study.

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

III. Extended Virtual Sensing Application

Vehicles, either human-driven or autonomous, can benefit from data coming from multiple sources, e.g., data generated by the dynamics of the 'ego' (i.e., the one under observation) vehicle, as well as onboard sensors such as advanced driver-assistance systems (ADAS) (e.g., ultrasonic detectors, lidar, radar, camera). Nevertheless, EVS services still play a pivotal role in providing high safety standards. Through vehicle-to-infrastructure (V2I) communication, an EVS server can receive the information collected through the aforementioned sources from the vehicles travelling over a given geographical area, combine them, and send the resulting information back to every vehicle as key input to its control logic. In other words, the EVS server can exploit its centralized view of the area and provide relevant data to the vehicles, which can be interpreted as *virtual* sensor measurements. The benefits are twofold. First, thanks to its centralized view, the EVS server can provide vehicles with data that are significantly richer than those locally gathered at the vehicles. Second, EVS services allow vehicles that are not equipped with a full-fledged set of ADAS sensors, to benefit from advanced safety services.

The EVS service we selected is intersection control, aiming at avoiding the risk of collisions between vehicles approaching an intersection. Such a service exploits the Cooperative Awareness Messages (CAMs) – or, the equivalent Basic Safety Messages (BSMs) in the SAE standard – that are periodically transmitted by vehicles and collected by a third-party entity in a database, the so-called Cooperative Information Manager (CIM). In particular, the intersection control application extracts from the collected CAMs information about position, heading, speed, and acceleration of all vehicles travelling across the monitored intersection. Such information is the input to another core component of the service, namely, the collision detector, designed to detect vehicles on collision course between each other or with an obstacle. The detector output triggers the Decentralised Environmental Notification Messages (DENM) Decider: a component generating DENMs to be transmitted to the vehicles involved in the detected collision. Such messages are meant to alert human drivers, or to enable autonomous vehicles to activate their emergency braking system. The whole service runs on one or more host machines in the MEC platform, implemented in the proximity of a wireless Point of Access (PoA), e.g., an LTE eNB, covering the intersection. Below, we provide further details on the selected EVS service, while its implementation is described in Section IV-D.

A. The Collision Detection algorithm

The core of the EVS service is the detection algorithm, presented in Algorithm 1 and built on the state-of-the-art trajectory algorithm we sketched in [6]. In our work, we focus on collisions between vehicles; however, since the algorithm is based on generic trajectories, it can be applied to any kind of entities that may happen to be on a collision course (e.g., vehicles and pedestrians). Also, for simplicity, Algorithm 1 does not highlight the dependency on acceleration, even though acceleration has been accounted for in the implementation of the EVS service.

The algorithm is run upon each new CAM message generation by a vehicle travelling across the monitored intersection. The collision detection algorithm requires as input (Line 0):

- position and speed of the vehicle transmitting the last CAM in the area of interest (denoted below as ego vehicle), respectively identified by the two vectors $\vec{p_0}$ and $\vec{s_0}$, where the latter also includes information on the heading;
- the latest CAMs sent by all other vehicles traveling in the area, which are stored in $\mathcal{V}$.

In Line 1, the set $\mathcal{Z}$ of entities with which the ego vehicle could collide is initialized and, in Line 2, the future position of the ego vehicle is evaluated for each future time instant. Then the algorithm computes the position of each vehicle $v \in \mathcal{V}$ that recently sent a CAM (Line 4) and the distance $\vec{d}(t)$ between v and the ego vehicle (Line 5).

Thanks to the computation of $\vec{d}(t)$, we are now aware of the distance between the ego vehicle and the generic vehicle v, at any time t. To reduce false positives, our algorithm aims at producing an alert only for imminent collisions, i.e., for those whose time to collision is below a given threshold $t2c$. Furthermore, we take into account the fact that the position of each vehicle available at the EVS application, i.e., $\vec{p_X}$, refers to the front bumper of the vehicle. To consider the actual space taken by real vehicles, our algorithm raises an alarm if the distance between two cars goes below a threshold $d2c > 0$.

Since we are interested in the minimum value of $D(t)$, in Line 7 we compute $t^\star$, defined as the time instant at which the distance between the two considered vehicles is minimum. If $t^\star < 0$, the two vehicles are getting farther apart, whereas, if $t^\star$ is greater than the threshold $t2c$, a collision between them is not considered as imminent. In both cases, no action is required (Line 8). If $t^\star$ is between 0 and $t2c$, Line 11 computes the minimum distance $d^\star$ at which the two vehicles will be at time $t^\star$. The algorithm compares $d^\star$ against the threshold $s2c$:

Algorithm 1 Collision detection pseudocode

Require: $\vec{p_0}, \vec{s_0}, \mathcal{V}$

1: $\mathcal{Z} \leftarrow \emptyset$
2: $\vec{p_0}(t) \leftarrow \vec{p_0} + \vec{s_0}t$
3: **for all** $v \in \mathcal{V}$ **do**
4: $\quad \vec{p_v}(t) \leftarrow \vec{p_v} + \vec{s_v} \cdot t$
5: $\quad \vec{d}(t) \leftarrow \vec{p_0}(t) - \vec{p_v}(t)$
6: $\quad D(t) := |\vec{d}(t)|^2 \leftarrow (\vec{s_0} - \vec{s_v}) \cdot (\vec{s_0} - \vec{s_v})t^2 + 2(\vec{p_0} - \vec{p_v}) \cdot \cdot (\vec{s_0} - \vec{s_v})t + (\vec{p_0} - \vec{p_v}) \cdot (\vec{p_0} - \vec{p_v})$
7: $\quad t^\star := t \colon \frac{\mathrm{d}}{\mathrm{d}t} D(t) = 0 \leftarrow \frac{-(\vec{p_0} - \vec{p_v}) \cdot (\vec{s_0} - \vec{s_v})}{|\vec{s_0} - \vec{s_v}|^2}$
8: $\quad$ **if** $t^\star < 0$ **or** $t^\star > t2c_t$ **then**
9: $\quad\quad$ **continue**
10: $\quad d^\star \leftarrow \sqrt{D(t^\star)}$
11: $\quad$ **if** $d^\star \leq s2c_t$ **then**
12: $\quad\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{v\}$
13: **return** $\mathcal{Z}$

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

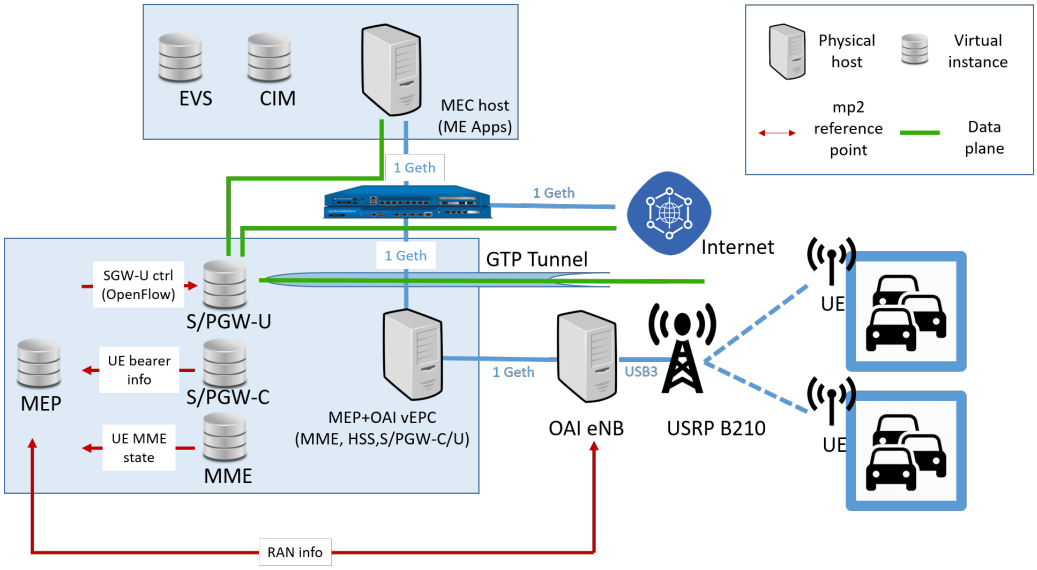

Fig. 2: Overview on the interaction between the test-bed building blocks

if $d^{\star}$ is lower, then vehicle c is added to set $\mathcal{Z}$, otherwise the algorithm skips to the next iteration of the cycle.

Once all CAMs in set $\mathcal{V}$ have been processed, the algorithm returns the set $\mathcal{Z}$ of vehicles with which the ego vehicle is on a collision course. If the set $\mathcal{Z}$ is empty, no action is taken, else an alert message is sent to the ego vehicle as well as to all those that are in set $\mathcal{Z}$.

IV. System Design and Implementation

We now illustrate the design and the implementation details of our MEC-based EVS application. The system includes four main blocks, namely, (i) the MEC-enabled Evolved Packet-Core (EPC) Network (Section IV-A); (ii) the procedures for service onboarding and instantiation within the MEC platform (Section IV-B); (iii) the vehicle emulator (Section IV-C); (iv) the EVS and the CIM services running as MEC applications (Section IV-D).

Fig. 2 provides an overview of the interactions between such building blocks. In the test-bed, two realistic implementations of cellular User Equipments (UEs), based on Open Air Interface (OAI), act as vehicles. Each UE periodically sends the information related to the position, speed, acceleration, and direction of several emulated vehicles towards a third party database, the CIM. In turn, the MEC-enabled EPC identifies the EVS traffic directed towards the CIM and applies traffic redirection to keep it at the edge. The EVS, which onboards the trajectory-based algorithm that detects approaching vehicles on a collision course, periodically retrieves the latest vehicle information received by the CIM. When needed, the EVS sends alerts towards the vehicles, exploiting again the same traffic redirection rules that the MEC-enabled EPC used for the uplink traffic.

A. A MEC platform based on OpenAirInterface

Our system builds on OpenAirInterface (OAI) [7], an open source implementation of a full LTE network, spanning the RAN and the EPC, with current developments focusing on 5G technology. On top of this, we have implemented our MEC platform, which exposes REST-based API endpoints to the MEO and ME applications, so that they can discover, register, and consume MEC services, including traffic redirection and, in our case, the EVS applications.

We provide extensions to the OAI RAN and the core network elements to implement the Mp2 reference points. Core network extensions are necessary for traffic offloading to ME application instances, while specific support is needed at the RAN level for retrieving radio network information from eNBs, such as per-UE channel quality indications (CQI), and exposing them to subscribing ME applications.

The Mp2 interface towards the RAN is implemented using the FlexRAN protocol [8], which is integrated into the standard OAI software distribution. For traffic management, we have adopted the Control and User Plane Separation (CUPS) paradigm introduced by the 3GPP [9]. CUPS proposes to separate the data- and control-plane functions at the S/P-GW level. The S/P-GW has been split into two entities: S/P-GW-C and S/P-GW-U (C for control plane; U for user plane). The former is in charge of managing the signalling in order to establish the user data plane, while the latter is in charge of forwarding the user plane data. In our implementation, the S/P-GW-U is based on a version of OpenVSwitch (OVS), patched to support GPRS Tunnelling Protocol (GTP) packet matching. When requested over its Mp1 interface, the MEP installs traffic rules on the S/P-GW-U to offload traffic to the MEC applications by remotely executing OpenFlow commands. The MEP needs to be aware of specific UE bearer information (UE IMSI, GTP tunnel endpoint identifiers, UE and eNB IP addresses) in order to appropriately install these rules. This information is available at the S/P-GW-C level upon UE bearer establishment, and we have modified the OAI EPC code to communicate it to the MEP via its REST Mp2 interface.

In our MEC testbed, ME applications are running on the MEC host as VMs directly on top of the kvm [10] hyper-

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

visor. However, our MEC platform is also compatible with VIMs such as OpenStack [11], while it has been tested with containerized ME applications managed by lxd [12].

Fig. 2 presents our MEC testbed setup and the interactions and interfaces between its components. The OAI EPC is virtualized, with the HSS, MME, and SPGW running as separate kvm VMs on a single physical machine, which also hosts the MEP. Note that the latter can also be executed as a virtual instance on the MEC host. Due to its real-time constraints, the OAI eNB software runs on a dedicated host, to which a USRP B210 RF board is attached.

B. Deployment of the service components as MEC applications

Next, we detail the procedures for the deployment of our automotive service components as MEC applications on our platform, from application preparation to instantiation and runtime management.

1) Application preparation and onboarding: According to the ETSI MEC standards, MEC applications are characterised by an application descriptor (AppD) [13], which is prepared by the service provider (in our case, the automotive vertical) as part of an application package. The AppD is the equivalent of a VNF descriptor (VNFD) in the ETSI NFV-MANO context [14]. It provides information necessary for application deployment, including a reference (URL) to the actual application image, application latency requirements, minimum requirements such as the amount of computing resources that should be allocated for an application instance, MEC services that the application exposes or consumes, and, importantly, DNS rules and traffic filters. The latter ones define the characteristics of the traffic that should be offloaded to the MEC application instance (e.g., traffic flows matching a specific protocol-destination and address-port tuple).

The vertical service provider submits an application package for *onboarding* to the OSS/BSS via the Customer Facing Service (CFS) Portal. The OSS/BSS then onboards the application package to the MEC system by communicating with the MEO over the standardized Mm1 reference point, thus making it available for instantiation.

In our case, the vertical service is composed of two main components (CIM and EVS), in turn broken down into specific modules that can be run as independent services. Our design does not preclude monolithic implementations, where all components are (i) provided by the same entity, (ii) bundled in a single package (software image) by the vertical, and (iii) deployed in a single, standalone, MEC application instance. However, this would limit deployment flexibility and scaling capabilities. Furthermore, we expect that in a real-world implementation the CIM will be provided by a different entity, such as a transportation authority, and would expose its information to other MEC applications (such as our EVS service). Therefore, we opted for a micro-services based implementation where each component is onboarded and instantiated separately. As detailed later, appropriate, standard-based service registration and discovery procedures are used so that the application components (including those aboard the vehicles) can discover and communicate with each other.

In conclusion, in our vertical service design, the following two MEC application packages should be provided:

- **CIM package**, integrating all the software related to the CIM, including both the CAM receiver and the Information Manager. The CIM features a modular design, with the CAM Receiver, Information Manager, and other components operating as separate, networked subservices. Since one CIM instance is expected to be active per monitored area, without significant scaling requirements, we chose to package all its modules into a single application.
- **EVS package**, including the EVS manager, the collision detection algorithm, and the DENM Decider. Due to its scaling requirements, the collision detection algorithm is deployed as a separate application sub-package, whose instances can be scaled in/out independently from the others, when needed. The EVS manager and the DENM Decider, instead, are built into a single application subpackage.

We will provide further details on these packages in Section IV-D.

2) Instantiation: After the application packages have been onboarded, as per the request of the vertical over the CFS portal, the OSS/BSS uses the *Application Lifecycle Management Interface* of the MEO (Mm1 reference point) to instantiate the CIM and the two EVS sub-packages. In each request, the identifier of the respective application package is included, according to the procedure described in ETSI MEC 010-2 [13]. The information that is included in the AppDs is used to configure the MEP for traffic redirection, update the DNS service of the MEC service, and register the necessary service API endpoints of each component with the MEP.

3) Service discovery: Regarding the CIM, the following requirements need to be satisfied:

- Since CIM virtual instances are created dynamically within the MEC system and since each client, depending on its location and the edge system covering it, needs to deliver its CAMs to the CIM instance covering its region, a mechanism has to be in place to *steer* CAMs to the appropriate MEC instance; this needs to take place transparently and with minimal UE involvement.
- The CIM service may be provided by a third party, such as a transportation authority, and a single CIM instance may have to provide its information to multiple EVS instances. Therefore, upon the deployment of a CIM MEC instance, its service endpoint needs to be registered with the MEP, so that it can be discovered by EVS applications.

To let vehicles discover the IP address of the CIM that collects their CAM messages, we use a combination of standard DNS mechanisms and MEC capabilities. We reasonably assume that the devices onboard vehicles are pre-programmed to search for the CIM at a well-known DNS name. To encode it, we use the `appDNSRule` field of the AppD. When the CIM application is instantiated, the MEO will instruct the MEP(M)

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

via the Mm3 reference point to update the MEC DNS database with an entry to resolve the CIM name to the IP address of the new MEC application instance.[1]

If, instead, the vehicular UEs are pre-programmed to communicate with a fixed CIM IP address (or receive this IP address from a centralized control entity), we add specific `appTrafficRule` entries in the AppD, so that appropriate traffic redirection rules can be set up in the MEC platform. In particular, this field allows specifying traffic filters that can match specific flows, identified among others by service IP address-port-protocol tuples. Upon service instantiation, the MEO extracts `appTrafficRules` from the AppD and communicates with the MEPM via the Mm3 interface to apply them. The MEPM, in turn, accesses the MEP's traffic rules service (in our implementation, over a REST interface), and the latter eventually applies them to the S/PGW-U over the Mp2 reference point. This type of traffic steering builds on SDN and is transparent to the UE: CAMs are sent to the well-known IP address and port of the CIM, and the S/PGW-U offloads the traffic to the IP address/port of the MEC instance by applying packet-rewriting OpenFlow rules installed by the MEP.

EVS instances, on the other hand, consume the CIM service via the Mp1 interface. When the CIM is instantiated, the MEO extracts the `appServiceProduced` field from the AppD. This field provides a description of the service endpoint exposed by the CIM, which the EVS Manager component needs to access to consume the input to the collision detection algorithm. Finally, the MEO adds the service to the MEP Service Registry.

4) Scaling: The EVS collision detection algorithm is executed under a stringent latency constraint, and its running time depends on the number of vehicles in communication range. Therefore, when this number increases so that the latency constraint cannot be met any longer, more resources need to be allocated to the algorithm. We address this situation by means of an *scaling out* operation, i.e., by creating one or more additional application instances, each assigned a different, smaller, coverage region, hence responsible for handling fewer CAMs. However, and contrary to VNFs deployed in an ETSI MANO environment, the ETSI MEC standard in its default, standalone version[2], does not directly support scaling, which is indeed an innovative feature in real-world platforms. To enabling scaling out while remaining compliant to the standard, we adopt the following approach.

It is typically envisioned that the vertical is offered an entry point to manage its application (e.g, a publicly accessible IP address to connect to the EVS) via the CFP. This gives the opportunity to execute application-specific management software, which monitors the number of vehicles per region and, when deemed appropriate, requests the instantiation of a new collision detection algorithm instance. This follows the standard MEC application instantiation procedure: the (external) monitoring component will request the creation of the instance via the CFS, which will convey the request over the Mx1 reference point to the OSS/BSS. The latter will finally communicate with the MEO over Mm1 to create the new instance[3].

We remark that this method uses only standard-based reference points and procedures. In our testbed implementation, we apply such an approach with just a small simplification: the decision to scale in/out is taken directly by the DENM Decider, which acts as an application-level orchestrator. This component monitors the number of vehicles in the region it controls, and communicates directly with the MEO to manage the instantiation or termination of MEC applications via the Mm1 functions.

C. Vehicle emulator

As mentioned, in our test-bed two UEs act as vehicles. The mobility traces describing the pattern of all emulated vehicles are obtained previously running the well known Simulation of Urban MObility (SUMO) [15] tool. We sample the mobility traces of each vehicle every 0.1 s (10 Hz) and we record key information of vehicle movements, such as position, speed, acceleration, and direction. For each obtained sample, we create a CAM, which is transmitted towards the eNB of the OAI cellular network.

The radio interface of the two UEs used in the test-bed exploits a standard OAI UE implementation. Each UE is emulated by a PC, equipped with an octa-core processor at 1.8 GHz and 16 GB RAM and connected to a USRP B210 RF board. Over-the-air communication is further improved by a pass-band filter, which reduces undesired interference at the receiver. Each UE also onboards the software for transmitting CAMs and receiving DENMs (the latter being alert messages sent by the EVS to the vehicles). This software, named VehicleSimulator, is a C++ standalone Linux application. Fig.3 illustrates its architecture and its main software components.

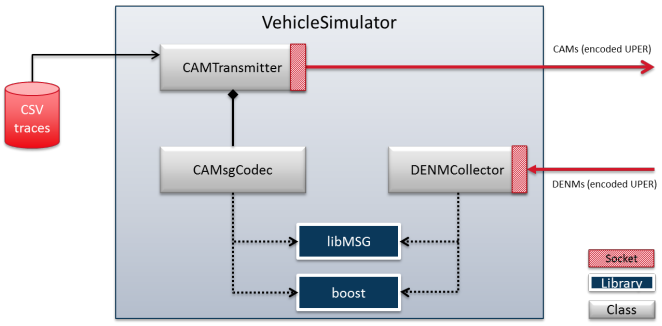

Fig. 3: Architecture of the VehicleSimulator

The CAMTransmitter is the main class of the VehicleSimulator. As the name suggests, the most important operation performed by the CAMTransmitter is the transmission of CAMs. In order to perform such an operation, the CAMTransmitter receives as input a CSV file, where each line, containing the transmission time and all other relevant information, represents

[1]This implies that vehicular UEs use the (local) MEC DNS to resolve the domain name of the CIM. This is a fair assumption, since the UE DNS server is assigned by the mobile network and MEC operator.

[2]I.e., not the MEC-in-NFV one.

[3]The collision algorithm application instance is already onboarded the first time the application is deployed at the edge.

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

a single CAM. For each CAM, the CAMTransmitter stores the corresponding CSV line in the RAM, in order to guarantee a faster access to the information, and it starts a new thread that manages the CAM creation. Such thread calls the CAMsgCodec class, which performs:

(i) the creation of the CAM structure, allocating memory to each CAM field. Such structure is obtained by the libMSG library of the open source CAM compiler ASN1 [16];
(ii) the parsing of the CSV line passed by the CAMTransmitter;
(iii) the update of the CAM structure with the received information;
(iv) the check of the consistency of the CAM structure with the standards [17];
(v) The encoding of the CAM structure with the Unaligned Packet Encoding Rule (UPER) to obtain the byte array to transmit.

When ready, the CAMTransmitter sends the CAM via UDP socket towards the eNB of the OAI cellular network and stores the time instant at which such a transmission happens. Note that the multi-thread structure of the CAMTransmitter allows the encoding of CAMs at the millisecond time-scale, hence the transmission of messages generated by multiple vehicles.

When a collision is detected, the EVS has to transmit a DENM towards the involved vehicles, i.e., towards the UEs where the vehicles are emulated. To send the CAMs to the correct UE, the EVS exploits the fact that the CIM stores also the IP address of each vehicle in the system. Within the VehicleSimulator, the DENMCollector listens to a specific port targeted by the EVS transmissions and decodes received DENMs. To guarantee better performance, the DENMCollector exploits two different threads. The first thread listens to the UDP socket used by the EVS and it performs the decoding and the storing of the DENMs in a dedicated queue in the RAM. The second, instead, processes the information in the queue where DENMs are stored and it records the time needed by the VehicleSimulator to decode each DENM. The boost C++ libraries have been used for socket operations and for the queue structure, which is optimized for the single producer/single consumer use case.

D. The automotive MEC service

We now provide more details on the two components of the automotive MEC service: (i) the CIM package, which acts as a collector of CAMs transmitted by the vehicles in the monitored area (Sec. IV-D1), and (ii) the EVS package onboarding a trajectory-based algorithm, which aims at detecting and notifying the risk of collision (Sec. IV-D2).

1) The CIM package: All CAMs originated by the two UEs are redirected by the MEC-enabled EPC towards the CIM. The CIM is an evolution of the Local Dynamic Map (LDM), a facility standardized by ETSI that maintains information influencing or influenced by road traffic and used to support Intelligent Transportation Systems (ITS). Data can be received from different sources, such as vehicles, infrastructure units, traffic centres, personal ITS stations, and onboard sensors and applications. The LDM offers mechanisms to grant secure access to the stored data. For example, it can provide information on the surrounding vehicles and Road Side Units (RSU) to any authorized application that requests it. Following the same concept, the CIM is a data storage located in a MEC host, able to receive all information relevant to the EVS application from vehicles traveling over a given area. As mentioned, the structure of the CIM can be divided into two main blocks: the CAM Receiver and the Information Manager. The CIM runs in a virtualized environment with 2 cores at 3.6 GHz and 4 GB RAM.

The CAM Receiver, a C++ standalone application, whose structure is represented in Fig. 4, performs the following main operations: (i) it receives the encoded UPER CAMs messages from the vehicles; (ii) it decodes the CAMs extracting the vehicle data; (iii) it feeds the Information Manager forwarding all the extracted data.

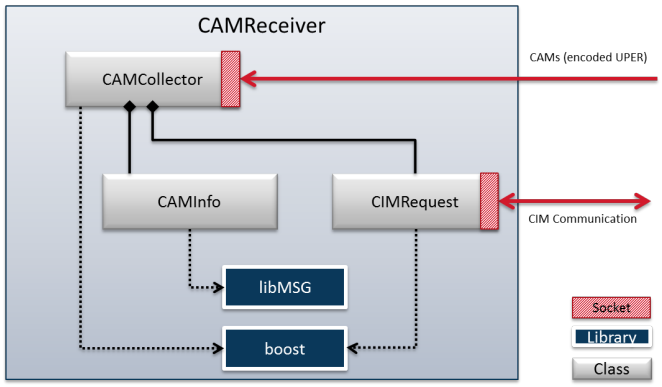

Fig. 4: The CAM Receiver architecture

The most important class of the CAM receiver is the CAM collector class. Such class receives over an UDP socket the CAM messages from the vehicles and, after decoding, it sends the received information to the Information Manager through the CIM Request object. Decoding of CAMs happens thanks to the CAM Info class. At each CAM reception, a new thread is started: a message is created and prepared for a UDP/TCP connection to the Information Manager and the computation time of the CAM reception operation is recorded.

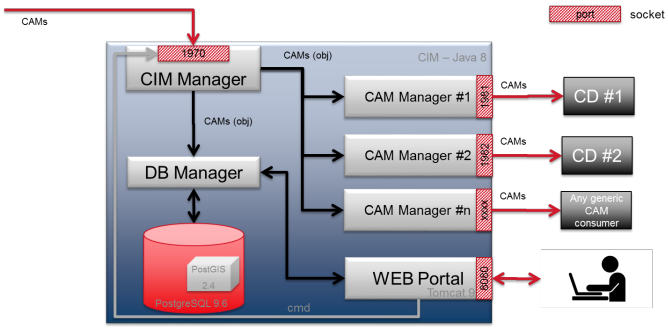

Fig. 5: The Information Manager architecture

The Information Manager, whose structure is shown in Fig. 5, is a standalone application with no User Interface

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

(UI), implemented as a JAVA 8 runnable JAR file. The UI is provided by a separated Web Portal implemented using Tomcat 9. The Information Manager presents one input port and an arbitrary number of output ports. The input port can manage multi-thread connections and is used to receive CAM messages from the CAM Receiver. The CIM can be configured to manage a particular circular area (defined by the latitude, longitude of the center, and by the radius of the circle). Only the CAM messages coming from vehicles in this area are handled, while all other messages are ignored. The input port is also used by the Web Portal for configuring the Information Manager.

The output ports can be created or disposed of from a previously defined list while the CIM is running. Each output port is managed by a module called CAM Manager that acts as an agent capable of satisfying the queries of a specific CAM consumer (i.e., a particular EVS instance). Each CAM manager receives a copy of all CAMs collected by the Information Manager from a module called CIM Manager. Furthermore, CAM managers are configured to manage a circular sub-area inside the monitored region (i.e. a crossroad under the control of a particular EVS instance) and only the subset of CAM messages inside such sub-areas is provided to the corresponding CAM consumers (see Fig.6).

The Information Manager can manage two storage mechanisms. The first one, which is used by CAM managers, allows the storage of the received CAM messages in the RAM memory (volatile mechanism), while the second one, which is optional, stores them in a PostgreSQL relational database (persistent mechanism). The volatile mechanism provides a faster access to the acquired CAM messages. As can be easily guessed, the amount of memory needed to store CAM messages is directly proportional to the number of active vehicles inside the area monitored by the CIM. Therefore, the volatile mechanism satisfies the requirement of readiness, which is of utmost importance for a collision detection algorithm, but it is not suitable for other needs, e.g., statistical analysis on all the messages acquired by the CIM. Using a PostGIS extension, the persistent storage mechanism can easily support statistical analysis through searches of messages inside geographical areas. These results can, for example, improve either the collision detection algorithm or the definition of the area under the algorithm control, i.e., the area under the CAM Manager control. In order to reduce the burden of the persistent storage mechanism on the performance of the Information Manager, a mechanism of multi-thread storage is implemented by the DB Manager module. Specifically, a queue for storage sessions is implemented and served when it is more convenient. Such an approach has the effect that persistent storage is performed with some delay and with the requirement of additional available memory.

2) The EVS package: The scope of the EVS package is to detect imminent collisions on a specific portion of the scenario under the control of the CIM. In our implementation, the EVS has been developed as a standalone C++ application, and it is deployed on a dedicated VM (1 core at 3.6 GHz and 2 GB RAM) in the MEC host. A summary of the structure of the EVS package is shown in Fig. 7.

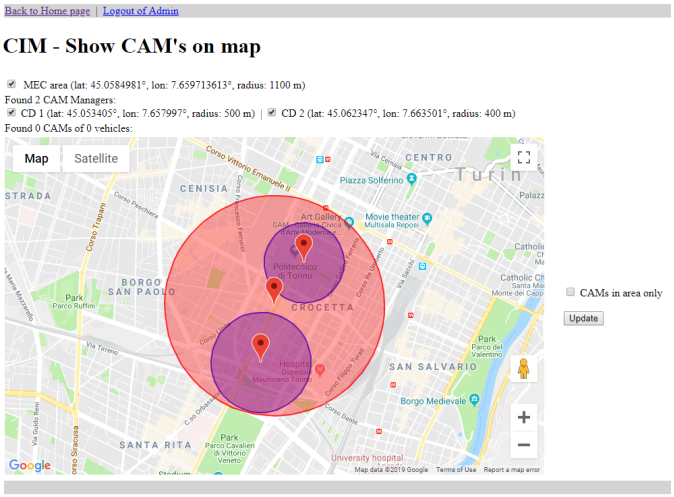

Fig. 6: An example of the map used for selecting the area monitored by the CAM manager on the CIM Web Portal

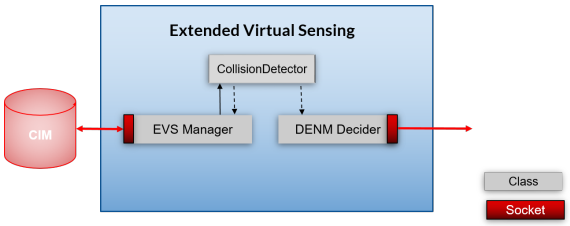

Fig. 7: Structure of the EVS package

The most important component of the EVS package is the so-called EVS Manager. The EVS manager can query a set of, not necessarily all, CAM Managers at the CIM, specifically, the ones covering the area of interest. Such implementation choice is motivated by the assumption that the CIM belongs to a transportation authority that does not run, but only responds to, the EVS application.

The EVS requests the latest CAMs to the CAM managers every 5 ms over a dedicated TCP connection. Such a threshold represents an optimal trade-off between the additional delay due to sequential queries to the CIM and the computational load of the EVS. CAMs are provided by the CIM in aggregate form in JSON format, with each CAM passed as a string field with a specific structure. The EVS manager interprets the response of the CIM and passes the new CAMs, if any, to the Collision Detector component. The Collision Detector updates the trajectories of the vehicles whose CAMs have just been received and compares them with the trajectories of all known vehicles in the area, as per Algorithm 1. In order to improve the system performance, not all the trajectories of vehicles are compared. In particular, if the distance between vehicles is larger than $2 \cdot v_{fast} \cdot t2c$ (where v_{fast} is the speed of the fastest vehicle between the two), then the check is skipped. Furthermore, to avoid considering the trajectories of vehicles with obsolete information, at this stage, the Collision Detector deletes from the memory all the vehicles whose last CAM was received earlier than 0.8 s.

For each pair of vehicles identified as on a collision course,

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

the Collision Detector creates a JSON with the relevant information and passes it to the DENM Decider. After the reception of the JSON data, the DENM Decider prepares and transmits unicast DENM messages to the involved vehicles. In order to avoid an excessive number of duplicated alerts, the DENM Decider does not generate the same DENM message for a given collision[4] more than twice every 100 ms. Such a feature of the DENM Decider allows implementing simple scale-out mechanisms for the EVS. In case the number of the vehicles in the monitored area increases and the processing time at the EVS does not meet the target latency constraint, the area of interest can be split and covered by two instances of the Collision Detector, instead of one. In this way, the number of vehicles under the responsibility of a Collision Detector instance decreases, as well as the processing time. Furthermore, collision detection may be ensured allowing an overlap region between the coverages of the two instances. As implemented, such scale out mechanism is sustained by the DENM Decider, since it can recognize and filter multiple notifications related to the same collision, even if coming from different Collision Detector instances.

The main classes of the DENM Decider are the DENManager and the DENMTransmitter. The former parses each JSON received by the Collision Detector and generates the values of the DENM fields; the latter manages the transmission of DENMs. The DENMTransmitter, in order to transmit a DENM, runs the DENMsgCodec class, the equivalent of the CAMsgCodec described in subsection IV-C. Thus, this class creates the DENM structure, updates it with the information contained in the JSON, checks the consistency with the standard, and finally encodes the DENM obtaining a byte array. After these operations, the DENM can be transmitted via UDP socket, through the eNB, and reaches the vehicle that risks colliding.

V. Performance Metrics

In this section, we present the different metrics we used in our experiments to assess the performance of the MEC system and of the two automotive MEC applications presented above. Specifically, we present latency related metrics (Sec. V-A) and metrics related to the ability of the EVS service to detect collisions (Sec. V-B). Using such metrics, we compare our MEC-based solution against a cloud-based implementation of the EVS, i.e., without exploiting the MEC traffic redirection rules and with the EVS and CIM packages running on a cloud datacenter (Sec. VI).

[4]A collision is identified by its location and the set of colliding cars.

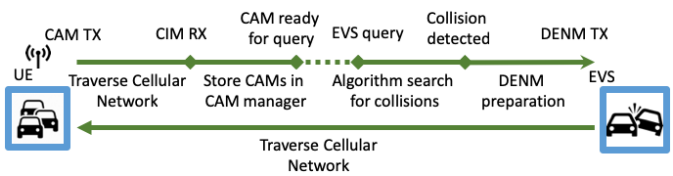

Fig. 8: End-to-end delay components

A. End-to-end delay and application processing times

To evaluate the ability of the EVS service to provide a vehicle on a collision course with real-time information, we use the end-to-end delay metric. The end-to-end delay is computed considering only CAMs that trigger an alarm, and it is defined as the time that elapses between the transmission of a CAM by a vehicle and the reception, by the same vehicle, of the alarm that such CAM triggered. I.e.,

$$D_{e2e} = T_n + T_c + T_w + T_e \quad (1)$$

where:

- T_n is the network latency due to the OAI network component;
- T_c is the time needed for the CAM Receiver to decode a CAM (encoded in UPER CAMs) and forward it toward the Information Manager, plus the time needed for the latter to store the message in its memory;
- T_w is the waiting time of the CAM, before being parsed by the EVS. Since every 5 ms the EVS queries the Information Manager, T_w may vary between 0 and 5 ms;
- T_e is composed of the EVS processing time and the DENM Decider processing time. The former is the time needed to query the Information Manager for the CAMs information fields, update internal tables with such information, run the collision detection algorithm, and create the JSON (for the DENM Decider) with the relevant information to create the DENMs. The DENM processing time is the time to receive the JSON from the EVS and create the DENMs for the vehicles involved in the detected collision.

Fig. 8 presents a summary of the above components. We remark that, while network delays strongly depend on whether the EVS/CIM packages run in the cloud or in the MEC, the processing time of the applications depends only on the computational resources assigned to them and on the incoming traffic that needs to be processed. Therefore, to profile the processing times of the EVS and the CIM, we also perform tests varying the number of vehicles in the scenario.

B. Collision detection

To evaluate the ability of our EVS implementation to detect imminent crashes in the area of interest, we first built a ground truth for the collisions. Thanks to the SUMO error-log file, for a specific mobility trace we obtain the actual collisions between vehicles if no EVS is implemented. Since the same mobility trace is also used in our test-bed, analyzing the DENMs correctly delivered in the test-bed, we can easily compute two fundamental performance metrics: (i) the percentage of detected collisions, and (ii) the percentage of false positives, i.e., the number of situations for which the Collision Detector triggers an alarm, but that do not lead to an actual collision. For both cases, i.e., collisions correctly detected and false positives, we present results to gain insight on the performance of our EVS.

Note that, if the alert for a collision was correctly transmitted, we look at when it was received and processed by the involved entities. In this way, we can determine if the

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

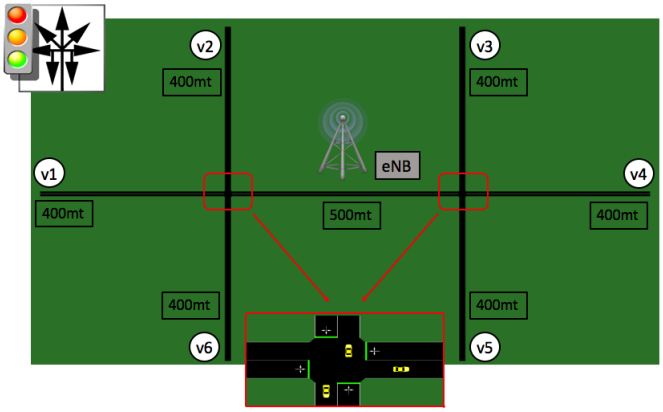

Fig. 9: Screenshot of the simulated scenario in SUMO

vehicle had sufficient time to brake before the impact. Whether a collision is detected in time or too late is determined in the post-processing phase by considering the final position where the involved vehicles stop. To do so, we retrieve the vehicles position at the moment of the first DENM reception. Then we focus on human-driven vehicles and account for the fact that a driver cannot start braking as soon as an alert message is received, for two different reasons: first, the vehicle human-to-machine interface (HMI) needs some time to elaborate the alert message; second, a human driver does not react immediately to an alarm. We set the HMI time to 400 ms, and the human reaction time to 1 s. Considering these two factors, we can compute the position of the front bumper when a car stops, given its current speed and maximum deceleration, hence assess whether the collision was detected in time or not.

If instead the alert turns out to be a false positive, i.e., an alarm is raised but no collision occurs, we compute the minimum distance between the trajectories of the two vehicles involved. In such a way, we can understand how far from an actual collision the involved vehicles were.

VI. Performance Evaluation

We now present the scenario we used for our performance evaluation (Sec. VI-A) and the obtained results, namely, end-to-end and processing latencies (Sec. VI-B), and collision detection performance (Sec. VI-C).

A. Reference Scenario

Two UEs emulate flows of vehicles traveling on the roads of the map shown in Fig.9. Vehicles traverse the scenario from north to south (or viceversa), and from east to west (or viceversa). Collisions happen only between vehicles crossing each other's path: no rear-end collisions are foreseen since we focus on the EVS service for collision avoidance at intersections. To simplify the DENM transmissions towards the pair of vehicles involved in a collision, we use one of the two UEs to emulate only vehicles in the north-south direction, while we use the second UE to emulate the presence of the vehicles travelling in the east-west direction. Finally, to evaluate how performance changes with the number of cars in the system, we consider three different values of vehicle density: (i) high, i.e., 20 vehicles/km, (ii) medium, i.e., 14 vehicles/km, and (iii) low, i.e., 7 vehicles/km. The inter-arrival times of vehicles into the system follows an exponential distribution, with mean set to the aforementioned values in the three different cases, respectively. For each vehicle density, we performed 5 different runs of 300 seconds each. We set the values of $t2c$ and $s2c$ to 3.5 s and 3.7 m, respectively, which, following the procedure outlined in [6], maximize the number of correctly detected collisions in our scenario.

B. End-to-end and processing latencies

As described in Sec. V-A, the end-to-end delay consists of three main components: (i) the network latency; (ii) the CIM storage and processing times; (iii) the EVS detection and DENM preparation times. Note that the only difference between our EVS MEC implementation and the equivalent EVS cloud implementation is represented by the network latency. In order to account for the additional delay required by the traffic to reach the CIM in a cloud server, we performed two measurement campaigns. With our OAI UE, we first pinged the CIM in the MEC 10,000 times and collected the experienced network latency. Then, to evaluate the effect of traversing a real cellular EPC to reach a cloud server, we used a commercial smartphone to ping the Amazon datacenter closest to Turin (the location of our test-bed), i.e., the one in Paris [18]. Fig.10 presents the cumulative distribution function (CDF) of the obtained network latency in the case of MEC and cloud-based implementations.

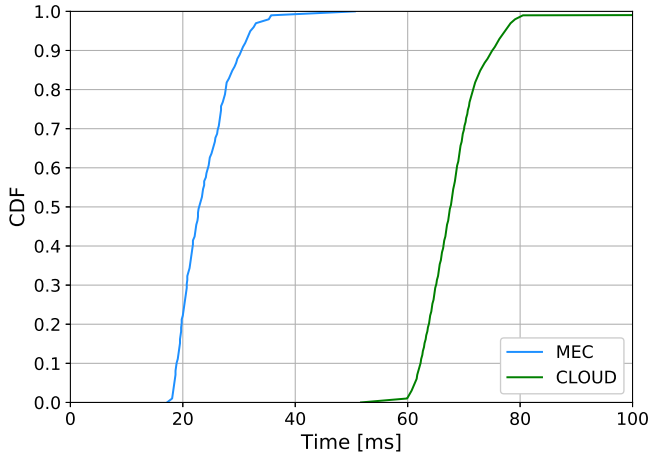

Fig. 10: CDF of the network latencies in the MEC and in the cloud experiments

Interestingly, the delay difference between cloud and MEC approximates a Gaussian distribution, with average 44.24 ms and standard deviation 8.36 ms. Thanks to this result, we use netem [19] for mimicking the effects of the cloud in our test-bed. Specifically, when evaluating the performance of the EVS application in the cloud, we introduce a Gaussian-distributed additional delay at the ingress port of the CIM and at the egress port of the EVS. The distribution of the total delay we added to each packet conforms to the measured difference between MEC and Cloud in our measurement campaigns.

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

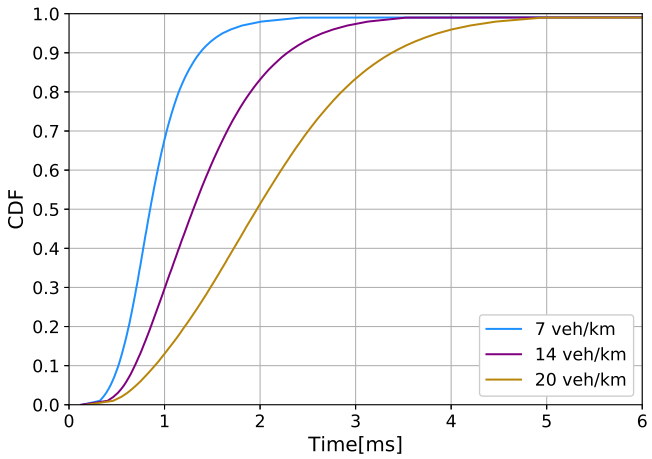

Fig. 11: CDF of the processing time of the EVS application

For what concerns the processing times, which statistically are the same in the cloud and the MEC experiments, we report in Fig.11 the CDF of the EVS processing time for the three vehicle densities we selected. Increasing the number of vehicles in the system clearly affects the performance of the EVS. Indeed, latency increases on average by 50% as we move from one vehicle density to the other. This is mainly due to the fact that the number of trajectory comparisons done at the EVS increases, and so does the processing time. Nevertheless, even for the highest value of vehicle density, in 99.99% of the cases, the EVS processes all CAMs and triggers all required alarms before querying again the CIM, i.e., within 5 ms.

Unlike the EVS processing times, the CIM processing times are barely affected by the increase of vehicles managed by the EVS application. Even in the case of high density, the worst-case processing time at the EVS is below 0.5 ms.

Finally, Fig.12 depicts the experimental CDF of the end-to-end latency of our MEC and cloud-based implementations. Each curve is obtained considering all DENMs received by the vehicles in the 5 different runs performed for each vehicle density. In our tests, the average number of collisions is 91 for the high-density case, 44.6 for the medium density, and 11.2 for the low density. Given the algorithm used and the parameters setting (i.e., the thresholds *t2c* and *s2c*), the maximum number of DENMs that can be generated for a given collision is 70. Indeed, in the best case, two cars on a collision course start receiving DENMs $t2c = 3.5$ s in advance, once for every CAM they transmit, i.e., two CAMs every 100 ms.

In both end-to-end latency distributions, there is a discrepancy between the summation of the four components we presented above and the total latency. Such discrepancy is mainly due to two reasons: (i) our EVS application does not query the CIM as soon as a CAM is logged into the corresponding CAM manager, but only once every 5 ms; (ii) the packets have to flow from a VM to another and such latency is not taken into account in any of the components we presented. On average, the observed discrepancy is 4.34 ms both for the cloud and the MEC implementation, which is in line with the latency we expect for the two components we cannot measure.

In all scenarios, in our MEC-based implementation, the 99.99% of the end-to-end latency values are below 50 ms. In particular, the average end-to-end latency is 29.55 ms for the low-density case, 29.89 ms for the medium density, and 30.5 ms for the high density. For our cloud-based implementation, instead, the end-to-end latency never drops below 50 ms: end-to-end latencies are on average 44 ms larger than the end-to-end latencies in the MEC-based implementation, which exactly corresponds to the network latency differences.

To understand if the end-to-end latency achieved by our implementation is good enough, we take as a reference the cycle time of LIDAR sensors aboard vehicles. LIDAR sensors typically refresh their information every 60 ms [20] and, for the information contained in the DENM to be coherent with on-board sensors, the maximum end-to-end latency should not exceed this value. As can be seen from Fig.12(left), our MEC implementation is well within the cycle time of a LIDAR sensor, even for the worst-case end-to-end latency in the high density case. On the contrary, the cloud-based implementation of the EVS application is constantly violating the 60 ms bound, meaning that the car may act upon obsolete information. As a matter of fact, recently automotive companies are leaning towards an even more stringent end-to-end latency for the EVS applications, i.e., 20 ms [21]. Such a latency is hardly achievable with 4G networks, even with the support of a MEC (see Fig.10), but it can be obtained using 5G cellular networks which, for critical applications, are expected to provide end-to-end latencies below 2 ms [22]. Given the fact that the total processing and communication times of our EVS and CIM VMs are under 10 ms in the worst case, we can conclude that our implementation is also consistent with such stringent latency requirement.

C. Collision detection performance

Thanks to the ground truth we built with the SUMO error-log, we now check if our EVS application can detect all occurring collisions. The result of our evaluation is that our MEC-based application can alert in time all vehicles on a collision course, and that all crashes are avoided, under all vehicle densities. On the contrary, the cloud-based implementation cannot detect on time two of the collisions that appear in the groud-truth trace under high vehicle density, for the reasons explained next.

Let us assume that our EVS application detects a collision at t^*. If the CAM used to detect the collision by the EVS was sent at $t_c \in [t^* - t2c, t^*]$, then a DENM is transmitted to the involved vehicles. For the collision to be detected on time, between t_c and t^*, we need the completion of a long list of events. Firstly, all actions from the reception of the triggering CAM to the delivery of the associated DENM (taking D_{e2e} time) must be executed; additionally, the processing of the received information by the vehicle and human-machine interface (taking D_h, namely, 400 ms) must be completed; the human driver's reaction (taking D_r, namely, 1 s) must then be taken into account; finally, the vehicle must be brought to a halt (in a time D_s, which, given the maximum vehicle speed

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

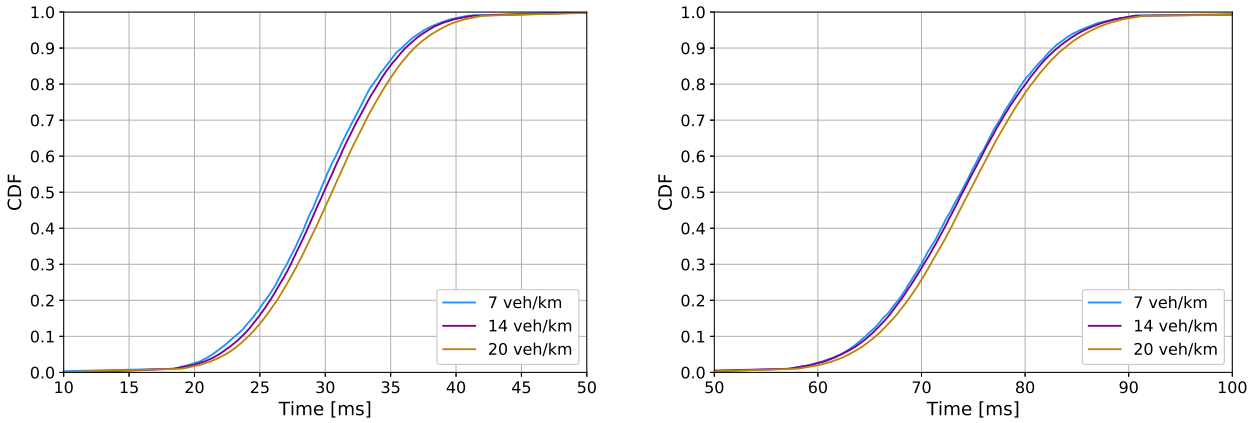

Fig. 12: CDF of the end-to-end delay for varying vehicle density, in the MEC-based (left) and in the cloud-based (right) implementation

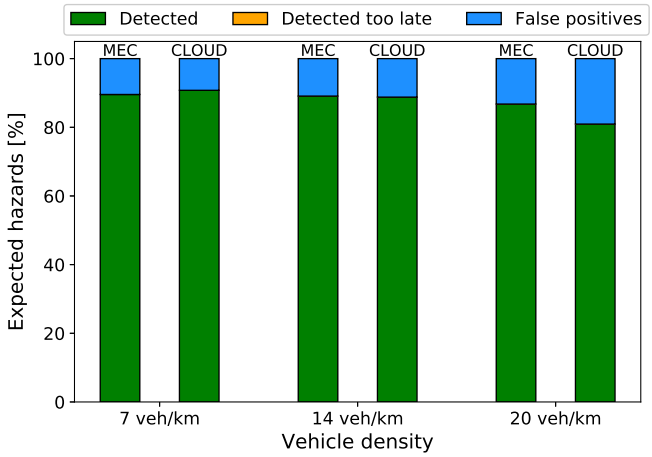

Fig. 13: Percentage of false positives: MEC vs. cloud

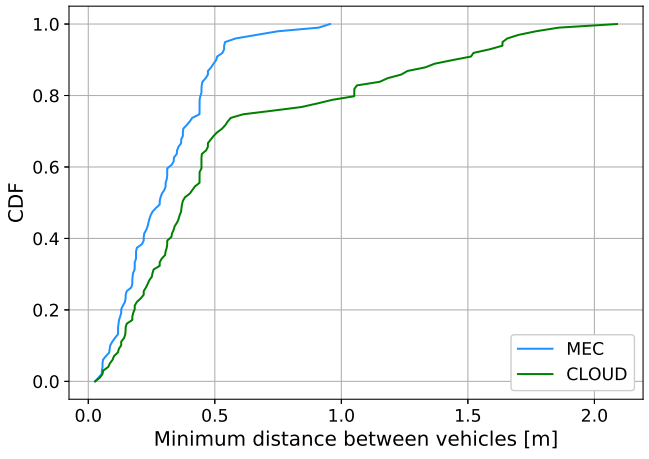

Fig. 14: Distances between cars involved in false positive detections: MEC vs. cloud

and deceleration in our settings, amounts to 1.85 s). Thus, we have:

$$D_{e2e} + D_h + D_r + D_s < t^* - t_c \leq t2c\,. \tag{2}$$

Additionally, by ETSI standard, CAMs are sent with a periodicity of 100 ms, thus, in the worst case, an offset as large as 100 ms should be considered for the CAM transmission. Given $t2c = 3.5$ s, it follows that D_{e2e} must be less than 150 ms. Looking at Fig. 12, both the MEC and the cloud implementation meet such requirement, however, for the MEC, in the 99.9% of the cases the EVS application can use at least two CAMs per vehicle for collision detection, even in the high-density case. This translates into more updated and reliable information on vehicle trajectories, as well as into a higher resilience to packet loss. The same cannot be said for the cloud implementation, where, with no losses, only one CAM is received by the EVS application within the above time limit, leading to the two missed collisions in our experiments.

Therefore, the larger network latency due to the cloud implementation impairs the ability of the EVS application to both achieve the reliability standards required for automotive safety services and integrate virtual sensing with traditional sensor measurements.

Finally, we look at the number of alarms unnecessarily raised by the EVS service. The percentage of false positives is almost as relevant as that of correctly detected collisions, since a large number of unnecessary alerts may affect the drivers' trust in the application. When we look at false positives, results are consistent with what presented above. Fig.13 shows the percentage of false positives obtained by the MEC and the cloud implementation of the EVS, while Fig.14 depicts the minimum distance between vehicles involved in situations that led to false positives. The additional latency suffered by the cloud version of our application causes a clear increase in the percentage of alerts directed toward vehicles that will not actually crash. Also, while in the MEC-based implementation the minimum distance between vehicles involved in false positive situations is always below 1 m, in the cloud-based implementation, such a value doubles.

In conclusions, our experiments show that the MEC is

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

undoubtedly the key enabler for delay sensitive applications, such as our EVS, while cloud-based implementations cannot meet the automotive ultra-low latency requirements.

VII. Related Work

Several works have dealt with applications in the automotive domain (e.g., [23]). Many of them, such as [24] and [25], propose collision avoidance and collision detection applications that do not leverage any mobile network infrastructure. In particular, [24] focuses on collisions between vehicles and pedestrians in industrial plants, with no specification of what type of wireless communication technology is used. [25] proposes instead a way to automatically detect a collision after it has occurred, using smart-phone accelerometers to reduce the time gap between the actual collision and the first aid dispatch. An automotive application leveraging mobile networks is [26], where vehicle collision detection is realized using solely vehicle-to-vehicle communications. Unlike our EVS framework, [26] relies on a continuously active direct communication between vehicles, which is not always available in urban environments.

Recently, cellular networks have emerged as a primary supporting infrastructure for the automotive domain. A considerable body of works, e.g., [27]–[29], perform comparisons between IEEE 802.11p and the standard LTE (non-V2V) network for vehicular applications. In particular, [27] highlights the higher capacity, coverage, and penetration of LTE with respect to 802.11p, which is also affected by scarce scalability and unreliable transmissions. [29] confirms these observations stating that LTE offers superior network capacity with respect to 802.11p and is suitable for all case studies. On the contrary, [28] considers LTE unsuitable for vehicle collision detection, due to the issues caused by the Doppler effect and LTE handoff procedures. The choice of the best communication technology is still the subject of an intense debate in the scientific community.

As far as MEC is concerned, an extensive body of works has studied MEC architectures (e.g., [30]) and developed analytical models, but concrete MEC system implementations are scarce. As an example, several theoretical works have addressed the design and dimensioning of MEC with the aim to maximize the amount of supported traffic [31], minimize service latency [32], or do both things for IoT applications [33]. Other works have addressed, again through analytical models, VNF placement within the MEC [34], as well as cloud and MEC-enabled access networks, see, e.g., [35], [36].

As far as experimental work is concerned, Subramanya et al. [37] present the design and implementation of a MEC platform with the goal of requiring no modifications at the RAN and EPC. Their approach is to maintain the necessary UE context information to carry out traffic steering by intercepting the S1 control plane traffic (S1-C) between the eNB and the MME, during UE attachment and handovers. CDS-MEC [38] also applies a similar approach to avoid any interaction between the EPC and the mobile edge system. However, we argue that currently it is not possible to have full MEC functionality in a way transparent to the network, for the following two reasons: (i) If S1-C traffic is encrypted, this approach does not directly work, since it is not possible to intercept the S1-C messages to monitor the necessary UE state at the MEC platform level; (ii) one of the main MEC platform services is the RNIS: without having an interface to the RAN (part of the Mp2 reference point), it is not possible to retrieve real-time radio network information from the eNBs, and this interface is not standardized. Therefore, we opted for a solution that requires a set of necessary extensions at the EPC and eNB levels to implement the Mp2 interface, which tailors our solution to OAI, particularly regarding the RAN part. Our work bears more similarities with LL-MEC [39], a MEC design also focused on OAI. As in our case, LL-MEC uses SDN techniques for control/user plane separation, as well as the same southbound protocol [8] to retrieve RAN-level information from OAI eNBs. Aside from other implementation differences (e.g., different Mp2 interface towards the EPC control/user planes), our MEC system further includes a standard-compliant implementation of the Mm1 reference point of the MEO towards the OSS/BSS, an RNIS interface that fully complies with ETSI MEC 012 [3], and platform components for MEC service discovery and registration. In summary, the following aspects and technologies developed in our work are totally original:

- MEP and its services and interfaces, in particular our Mp2 interface towards the EPC, are different from those used in other works [37]–[39], and the service registration and discovery features of our MEC platform are novel;
- MEO exposing the Mm1 interface to the OSS/BSS, as specified in ETSI MEC 010-2 [13]. This is a critical interface, since it is the entry point to the MEC system, allowing for MEC application onboarding, instantiation, and life-cycle management;
- the overall EVS service, collision detection algorithm, as well as our proposed mechanisms to support service scaling at the mobile edge.

Finally, we mention that a preliminary version of this work has been presented in our conference paper [40].

VIII. Conclusion

The provision on a global scale of low-latency communication services to vehicular applications will soon be realized by the deployment of 5G networks and the exploitation of edge infrastructure such as MEC. Such a technological enabler will be a key in the development of sophisticated road safety applications, exploiting the heightened awareness coming not only from on-board sensors, but also from information provided by the mobile infrastructure about what other vehicles in the surroundings are doing. One example of such an application is intersection control, aiming at reducing the risk of vehicle collision. In this paper, we have presented a MEC-based architecture and its test-bed implementation, supporting an Extended Virtual Sensing system and implementing a collision detection algorithm that leverages vehicular-to-infrastructure communication to alert drivers of potential, impending crashes. A hardware-in-the-loop simulation campaign has allowed us to identify the deployment parameters and it has shown

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2931878, IEEE Transactions on Network and Service Management

the effectiveness of our approach in avoiding collisions and stopping vehicles well clear of their potential crash.

ACKNOWLEDGMENTS

This work was supported by the European Commission through the H2020 5G-TRANSFORMER project (Project ID 761536). The work of Christian Vitale was also supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 739551 (KIOS CoE) and from the Republic of Cyprus through the Directorate General for European Programmes, Coordination, and Development.

REFERENCES

[1] F. Giust, X. Costa-Perez, and A. Reznik, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," *IEEE 5G Tech Focus*, vol. 1, no. 4, Dec. 2017.

[2] *Mobile Edge Computing (MEC); Framework and Reference Architecture*, ETSI Group Specification MEC 003, Jan. 2019.

[3] *Mobile Edge Computing (MEC); Radio Network Information API*, ETSI Group Specification MEC 012, Jul. 2017.

[4] *Multi-access Edge Computing (MEC); Support for Network Slicing*, ETSI Std. MEC 024, Jul. 2018.

[5] *Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NVF environment*, ETSI Group Report MEC 017, Feb. 2018.

[6] M. Malinverno, G. Avino, F. Malandrino, C. Casetti, C.-F. Chiasserini, and S. Scarpina, "Performance analysis of C-V2I-based automotive collision avoidance," in *IEEE WoWMoM*, Washington D.C., 2018.

[7] "OpenAirInterface, 5G software alliance for democratising wireless innovation," http://www.openairinterface.org, accessed: 2018-14-12.

[8] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. P. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," in *ACM CoNEXT*, 2016.

[9] P. Schmitt, B. Landais, and F. Y. Yang, "Control and User Plane Separation of EPC nodes (CUPS)," 3GPP, Tech. Rep., Jul. 2018. [Online]. Available: http://www.3gpp.org/cups

[10] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Linux Symposium*, 2007.

[11] "Openstack webpage," https://www.openstack.org/, 2019, [Online; accessed 10-March-2019].

[12] "Linux container 1xd webpage," https://linuxcontainers.org/lxd/, 2019, [Online; accessed 10-March-2019].

[13] *Mobile Edge Computing (MEC); Mobile Edge Management; Part 2: Application lifecycle, rules and requirements management*, ETSI Group Specification MEC 010-2, 2017.

[14] *Network Functions Virtualisation (NFV); Management and Orchestration*, ETSI Group Specification NFV-MAN 001, Dec. 2014.

[15] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility," in *International Conference on Advances in System Simulation (SIMUL)*, vol. 42, Barcelona, Spain, 2011.

[16] "Open source ASN1 compiler," http://lionet.info/asn1c/compiler.html, 2019, [Online; accessed 23-March-2019].

[17] I. T. S. I. ETSI EN 302 637-2 V1.3.1 (2014-09), "Vehicular communications; basic set of applications."

[18] "Netem," https://calculator.s3.amazonaws.com/index.html, 2019, [Online; accessed 18-April-2019].

[19] "Network emulation (netem) linux tool," https://wiki.linuxfoundation.org/networking/netem, 2019, [Online; accessed 18-April-2019].

[20] "Commercial lidar sensor for vehicle," https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-systems/predictive-emergency-braking-system/mid-range-radar-sensor-(mrr)/, 2019, [Online; accessed 18-April-2019].

[21] C. Casetti, C. F. Chiasserini, N. Molner, J. Martín-Pérez, T. Deiss, C.-T. Phan, F. Messaoudi, G. Landi, and J. B. Baranzano, "Arbitration among vertical services," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2018, pp. 153–157.

[22] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, pp. 1–125, 2015.

[23] L. Gallo and J. Haerri, "Unsupervised long-term evolution device-to-device: A case study for safety-critical v2x communications," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 69–77, 2017.

[24] Z. Riaz, D. Edwards, and A. Thorpe, "SightSafety: A hybrid information and communication technology system for reducing vehicle/pedestrian collisions," *Elsevier Automation in construction*, 2006.

[25] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "Wreckwatch: Automatic traffic accident detection and notification with smartphones," *Springer Mobile Networks and Applications*.

[26] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Cooperative collision avoidance at intersections: Algorithms and experiments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1162–1175, 2013.

[27] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro, "Lte for vehicular networking: a survey," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 148–157, 2013.

[28] Z. Xu, X. Li, X. Zhao, M. H. Zang, and Z. Wang, "DSRC versus 4G-LTE for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, 2017.

[29] Z. Hameed Mir and F. Filali, "Lte and ieee 802.11p for vehicular networking: a performance evaluation," *EURASIP Journal on Wireless Communications and Networking*, p. 89, 2014.

[30] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[31] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, 2018.

[32] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, March 2018.

[33] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in IoT networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.

[34] P. Zhao and G. Dán, "A benders decomposition approach for resilient placement of virtual process control functions in mobile edge clouds," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1460–1472, Dec 2018.

[35] B. P. Rimal, D. Pham Van, and M. Maier, "Mobile-edge computing versus centralized cloud computing over a converged FiWi access network," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 498–513, Sep. 2017.

[36] Y. Lin, Y. Lai, J. Huang, and H. Chien, "Three-tier capacity and traffic allocation for core, edges, and devices for mobile edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 923–933, Sep. 2018.

[37] T. Subramanya, G. Baggio, and R. Riggio, "lightMEC: a vendor-agnostic platform for multi-access edge computing," in *International Conference on Network and Service Management (CNSM)*, 2018.

[38] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun, "CDS-MEC: NFV/SDN-based application management for MEC in 5G systems," *Computer Networks*, vol. 135, pp. 96–107, 2018.

[39] N. Nikaein, X. Vasilakos, and A. Huang, "LL-MEC: enabling low latency edge applications," in *IEEE CloudNet*, 2018.

[40] G. Avino, M. Giordanino, P. Frangoudis, C. Vitale, C. Casetti, C. F. Chiasserini, K. Gebru, A. Ksentini, and A. Stojanovic, "A MEC-based Extended Virtual Sensing for Automotive Services," in *submitted to AEIT AUTOMOTIVE*, Torino, Italy, 2019.

1932-4537 (c) 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.